論 文
57-1-22

소프트에러 결함 허용 캐쉬

Fault Tolerant Cache for Soft Error

李 鍾 豪† · 趙 浚 東* · 表 正 烈** · 朴 基 豪**
(Jong-Ho Lee · Jun-Dong Cho · Jung-Yul Pyo · Gi-Ho Park)

Abstract - In this paper, we propose a new cache structure for effective error correction of soft error. We added check bit and SEEB(soft error evaluation block) to evaluate the status of cache line. The SEEB stores result of parity check into the two-bit shit register and set the check bit to '1' when parity check fails twice in the same cache line. In this case the line where parity check fails twice is treated as a vulnerable to soft error. When the data is filled into the cache, the new replacement algorithm is suggested that it can only use the valid block determined by SEEB. This structure prohibits the vulnerable line from being used and contributes to efficient use of cache by the reuse of line where parity check fails only once can be reused. We tried to minimize the side effect of the proposed cache and the experimental results, using SPEC2000 benchmark, showed 3% degradation in hit rate, 15% timing overhead because of parity logic and 2.7% area overhead. But it can be considered as trivial for SEEB because almost tolerant design inevitably adopt this parity method even if there are some overhead. And if only parity logic is used then it can have 5% ~10% advantage than ECC logic. By using this proposed cache, the system will be protected from the threat of soft error in cache and the hit rate can be maintained to the level without soft error in the cache.

Key Words : Cache, Tolerant, Parity, Soft Error, SEEB

1. 서 론

1.1. Soft error의 원인 및 영향

Soft error는 transient error[1], single event upset(SEU)[2], single event effects(SEE)[3], single event transient(SET)[4]등의 총칭이며 특정 환경이나 조건에 따라 발생하는 예측 불가능한 error를 일컫는다. Soft error는 이러한 비정규적인 특성 때문에 최고의 신뢰도를 가져야만 하는 자동차, 항공, 의료기기 등의 시스템에서 큰 문제를 야기시킬 수 있는데 예측할 수 없는 한번의 오류가 치명적인 결과를 가져 올 수 있는 것이다. Soft error의 원인은 1970년대에 밝혀진 alpha particle 외에도cosmic ray, signal integrity, random noise등이 있으며 이중 cosmic ray에 의한 error가 가장 큰 비중을 차지한다.[5] Cosmic ray에 의한 error는 지역 및 고도에 의한 영향을 받으며 비행기 안에서는 지면에서보다 최대 800배까지 위험이 증가한다. 또한 이 같은 외부적인 환경요인만이 아니라 soft error 에 취약한 design상의 구조적 결함이나 제조 공정의 영향도 soft error의 원인이 될 수 있다.[6] Soft error의 발생 빈도수는 SER(Soft Error Rate)로 나타내는데 단위로는 FIT(Failure In Time)이나MTTF(Mean Time To Failure)가 사용된다. FIT는 109시간 동안 발생한 soft error의 빈도수를 나타내며 MTTF는 발생 fail간의 평균 시간이다. 일반적으로1000 FIT는 114 year MTTT를 나타낸다. SER은 제품에 따라 많은 차이가 발생하는데 위의 예처럼(114 year) 제품의 life cycle을 넘어서는 것이 있는가 하면 단 몇 시간 내에 soft error 발생가능성이 있는 것도 존재한다. 이러한 design의 특성과 지역적 특성이 결합되게 되면 상상 할 수 없을 만큼 error의 위협은 증폭되어 시스템의 오동작을 유발 시킨다. Soft error의 영향으로 user가 인지하지 못하는 사이 시스템은 오동작 하게 되는데 갑작스런 시스템 reset, abort등의 예외상황 발생이 그것이다. 따라서 이러한 error가 발생했을 때 시스템 자체적으로 복구 할 수가 없다면 신뢰도에 큰 타격을 입게 된다. Soft error는 일반 로직 보다는 storage cell에 더 큰 영향을 주는데[7] 오류가 발생한 채로 저장되어 나중에 시스템을 무력화시키기 때문이다. 캐쉬 메모리에서의soft error 발생 사례가 이러한 상황을 잘 설명하여주고 있으며 이에 대한 해결 방안이 최근까지도 큰 이슈가 되고 있다. Soft error의 발생 가능성을 논할 때 사용되는 척도로는 Qcrit이 있다.[8] 가속된 alpha particle이나 neutron에 의해 발생한 전하 쌍에 의해서 cell의 상태가 반전되지 않으려면 Qcrit 이상의 전하는 보유하고 있어야 한다. Cell의 전하량이 Qcrit 보다 작다면 SEU가 발생할 수 있는 가능성이 큰 것으로 간주 된다. 공정이 미세해 질수록 내부 capacitance와 인가전압이 낮아지므로 Qcrit은 낮아지게 되어 soft error에

† 교신저자, 正會員 : 성균관대 정보통신 공학부 석사과정
E-mail : json.lee@samsung.com
* 非 會 員 : 성균관대 교수, 공학박사
** 非 會 員 : 삼성전자 SYTEM LSI 사업부 연구원
接受日字 : 2007年 10月 25日
最終完了 : 2007年 11月 9日

더욱 민감해 질 수 밖에 없다. 따라서 제조 공정의 미세한 이상에도 design내부적으로 soft error에 취약한 node가 발생 할 가능성이 커진 것이다. 이를 방지하기 위한 design 가이드가 있으나[9] 이러한 방법들은 capacitance를 늘리고 인가전압도 높이는 방법이어서 미세공정 개발로 인한 성능향상에 역행하는 결과를 가져온다. 왜냐하면 이러한 방법은 delay및 power를 증가시키게 되어 device의 성능을 떨어뜨리게 되기 때문이다. 따라서 전체적인 design의 topology를 바꾸기 보다는 soft error에 대한 시스템의 오 동작을 방지할 수 있는 장치를 마련하는 것이 시대적인 요구에 부합하는 것으로 판단된다.

1.2 Related work

캐쉬에서 발생하는 soft error를 해결하기 위한 노력은 최근까지 계속되고 있는데 그 원류는 캐쉬를 사용한 design에서의 수율 향상에 관한 연구에서 에서 찾아 볼 수 있다.[6][9][12] 이러한 연구에서 제안된 방식은 양산test 결과 hard defect으로 판명된 address의 캐쉬를 영구히 사용 제한하는 것이며 해당 address를 사용 금지시키기 위해서 캐쉬의 address decoder를 조작 할 수 있도록 구현 되어있다. 이러한 연구는 계속 되고 있으며 그 복잡도 또한 증가하고 있는 추세이다.[13][14] 또한 리얼타임으로 hard defect을 해결하기 위한 방안도 제시 되었는데 패리티 로직과 캐쉬의 동작 원리를 접목 시킨 방법이다. 패리티 검사결과 fail이 발생한 캐쉬 line을 추가적인 검사 비트를 통하여 표시하여 두었다가 차후에는 사용 제한하는 방법이다.[6] 패리티 검사 결과 fail이 발생하게 되면 외부 메모리에서 다시 data를 캐쉬 메모리로 읽어 오게 되는데 이때 다시 오류가 발생하면 hard defect로 간주 하여 사용 제한하는 것이다. 위에서 언급된 모든 내용은 soft error를 해결하기 위한 방안에 응용될 수 있다. 왜냐하면 soft error도 순간으로 볼 때에는 defect로 간주 될 수 있기 때문이다. 따라서 이러한 defect에 대한 fault tolerant 구조에 soft error의 특징인 비정규성을 추가 하게 되면 soft error tolerant design이라 할 수 있겠다. Soft error tolerant 구조를 크게 두 가지로 나누어 보면 ECC(Error Correction Code)에 의한 방법과[15] 위에서 언급된 패리티 로직과 캐쉬 시스템을 사용하는 방법이다. ECC에 의한 방법은 단일 비트 error에 대해서는 오류 복구 성능이 좋지만 overhead가 커서 사용에 신중을 기해야 한다. 통상 ECC에 의해 발생하는 timing overhead는 단순 패리티 검사 방식 보다 5%~ 10% 크기 때문에[16][6] 패리티와 캐쉬 시스템을 이용한 방법을 응용하면 간단하면서 성능면에서 이점을 가질 수 있다. 이 방법에서는 패리티 로직을 추가하여 해당 캐쉬 line의 오류 여부를 판단하도록 되어 있으며 오류가 발생한 line은 무효화 시켜서 재사용 하도록 구현 되어 있다.[16]. 하지만 패리티 검사에서error가 발생한 line은 1.1절에서 언급된 바와 같이 soft error에 취약한 구조를 가지고 있을 가능성이 크다. 따라서soft error에 취약한 구조인지 판별한 후 해당 line을 재사용 할 것인지 결정하는 절차가 요구된다. 따라서 본 논문에서는 soft error 발생 시 캐쉬 line 의 사용제한 방법과 해당 line이 soft error에 취약한지 판별하는 로직 및 이 경우의 linefill 방안을 제안하고자 한다.

2. 본 론

2.1 일반적인 캐쉬구조

그림 1은 일반적인 캐쉬 구조를 보여주고 있다. Address는 tag, set, word의 세 부분으로 나뉘어 진다. Tag는 access 하고자 하는 address의 일정 부분으로서 linefill시 data와 함께 저장된다. 특정 address내에 저장되어 있는 data를 참조 하고자 할 때에는 함께 저장하고 있던 tag 내용과 access하려는 현재 address의 tag 부분이 일치하는지 확인 하여 동일하면 hit, 다르면 miss가 발생하도록 되어 있다. Set은 캐쉬 메모리에서의 주소를 나타내며 index라고도 한다.

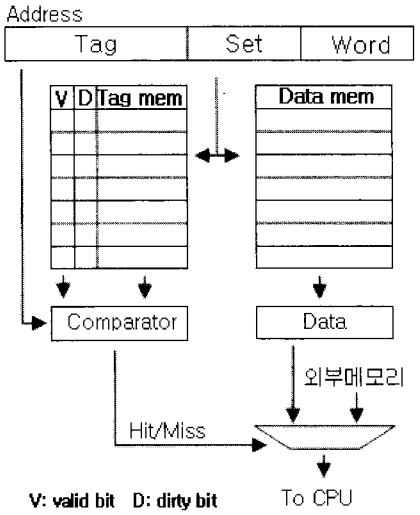

그림 1. 기존 캐쉬 구조
Fig 1. Existing cache structure

캐쉬는 구성에 따라 두 종류로 나눌 수 있는데 set associative 캐쉬의 경우는 동일한 index주소가 way 마다 존재하며 direct mapped 캐쉬인 경우는 이러한 way가 한 개인 경우이다. 이러한 way는 동일한 index의 data가 여러 곳에 존재 할 수 있도록 하여 캐쉬의 hit ratio를 향상시켜 준다. 외부 메모리로부터 캐쉬에 data를 가져다 놓는 동작을 linefill이라 한다. Data는 line이라는 단위로 캐쉬에 linefill되며 line은 다수의 word로 이루어 져 있다. 한 예로 ARM사의 ARM920T core인 경우는 8개의 word가 1line을 이루고 있다. CPU는 hit가 발생하면 캐쉬의 내용을 miss가 발생하면 외부메모리의 내용을 참조하게 된다.[17]

2.2. 기존의 soft error tolerable 캐쉬 구조

그림 2는 기존에 제안 되었던 tolerant 캐쉬 구조 이다. 일반 적인 캐쉬 구조에 패리티 관련 로직이 추가 되어 있다.[18] 기본 적인 동작은 패리티 검사 error가 발생 하였을 때 해당 line을 무효화 시키는 것이다. 이러한 방법으로 캐쉬 line상에 soft error가 발생 하였을 때 외부 메모리로부터 다시 data를 가져오는 메커니즘을 구성하고 있다. 이러한 구성의 문제점은 간단하게 soft error를 복구 할 수 있기는 하나 design의 구조적인 취약점은 여전히 존재할 수 있다는 것이다. 이러한 문제점은 1.2절에서 언급된 hard defect에서

의 캐쉬 line 사용제한 방법으로 어느 정도 해결 될 수 있다. 하지만 error가 발생한 line을 무조건 사용제한 한다면 장기적인 운용면에서 캐쉬 메모리 의 성능 저하를 가져 올 수 있다. 따라서 MTTF가 비교적 긴 design에서는 soft error가 발생한line에 대해서 다시 한 번 취약성을 검토하여 line의 재 사용 여부를 결정하는 구조가 요구된다.

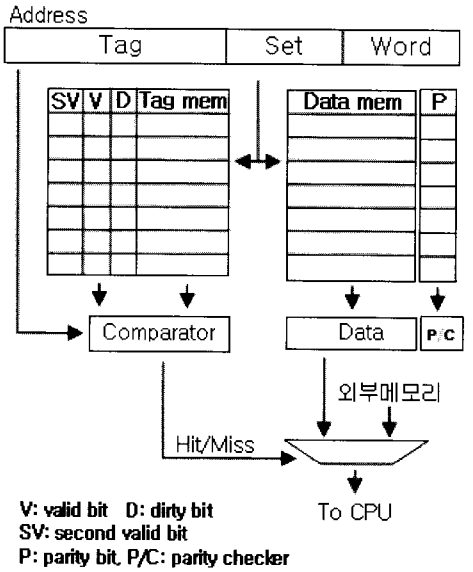

그림 2. 기존의 tolerant cache 구조
Fig 2. Existing tolerant cache structure

2.3. 제안된 캐쉬 구조

그림 3처럼 제안된 캐쉬 구조에는 일반적인 캐쉬 구조에 SV 비트(second valid bit), SS 비트(soft error suspicious bit), SEEB(soft error evaluation block), 패리티 비트, 패리티 checker, 수정된 replacement counter 그리고 hit/miss 결정 로직이 추가 된다. SV 비트는 기존 논문을 참조 하였고 패리티 관련 로직은 이미 널리 쓰이고 있으므로 따로 설명이 필요 없을 것으로 생각된다. 그리고 새로 제안된SS 비트, SEEB, hit/miss 결정 로직, replacement counter등의 각각 로직은 아래의 캐쉬 동작 설명 시 구체적으로 다루기로 하겠다.

2.3.1 Compulsory linefill시의 동작

Compulsory linefill은 캐쉬에 없는 data를 외부메모리에서 처음 가져오는 것이다. 이때access하려는 address의 tag 및 외부 메모리로부터 가져온data를 캐쉬의 tag와 data ram에 저장하며 valid 비트를 1로 수정한다. Reset후 처음 사용하는 line은valid 비트가 0이고 SV 비트는 1이다. 패리티 checker는 valid 비트가 0 인 경우에는 동작하지 않도록 구현되어 있으며 따라서 compulsory miss가 발생하면 바로 linefill이 수행된다. SV 비트는 second valid 비트이며 캐쉬 line이 유효한지를 나타낸다는 점에서 valid 비트와 비슷하다. 기존 논문에서는 각각 available bit[19], purge bit[6], second valid bit[20] 처럼 다른 이름으로 명명 되었지만 모두 line의 유효성을 나타낸다는 점에서 기능은 동일하다. 본 논문에서의 SV 비트는 패리티 검사 결과를 저장한다는 것이 valid 비트와 다른 점이다. 본 논문에서 사용되는 SV 비트는 해당 line이 패리티 검사에서 2번 이상 fail이 발생하였을 때에 0으로 설정 되며 해당 line을 사용 제한시키는데 사용된다. 또한 SV 비트는 캐쉬 읽기 시에 hit/miss를 결정하기 위한 중요한 요소가 되는데 자세한 내용은 2.3.2절에서 다루도록 하겠다. Reset 이후에는 SV 비트가 1로 초기화 되는데 이것은 초기 조건에서는 모든 line이 정상이라고 가정한다는 의미이다. 따라서 compulsory linefill은 SV 비트의 영향 없이 이루어진다.

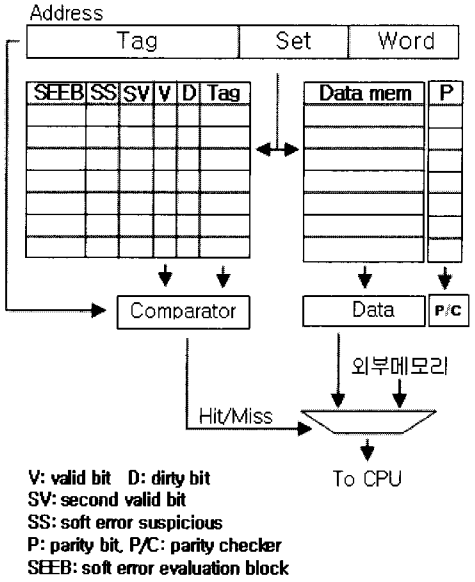

그림 3. 제안된 캐쉬 구조
Fig 3. Proposed cache structure

2.3.2 캐쉬 읽기 시의 동작

캐쉬에 저장된 data를 다시 CPU에서 요청을 하였을 때 비로소 soft error tolerant 기능은 동작하게 된다. hit/miss 로직은 access하려는 address의 tag 부분과 tag 메모리의 내용을 비교하고 data에 대한 패리티 검사를 수행 하는 동시에 SV 비트를 확인 하여 해당 line이 유효한지(SV 비트가 1) 확인 하여야 한다. Tag 비교에서 hit가 발생하여도 SV 비트가 0이면 결과적으로 miss가 발생하며 외부 메모리를 access하게 된다. SV 비트가 0이라는 것은, 2.3.3절에서

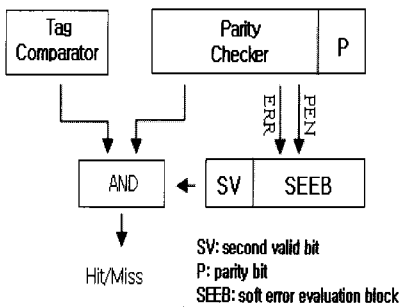

그림 4. 캐쉬 읽기시의 동작
Fig 4. Cache read operation

다루어질 내용이지만, 이미 soft error가 2회 발생하여 사용제한이 된 line이라는 의미이기 때문이다. SV 비트를 제어

하는 블록을 SEEB(soft error evaluation block)라고 하는데 이 블록에서는 soft error에 취약하다고 판단된 line의 SV 비트를 0으로 설정하도록 되어 있다. SEEB는 전체 way의 모든 index에 삽입되어 있으며 패리티 검사의 결과를 저장하는 역할을 한다. 또한 저장된 패리티 검사 history를 가지고 해당 line을 soft error free line(SF), soft error 의심 line(SS) 그리고soft error 발생 line(SE)으로 분류한다. 해당 line이 SF이거나 SS 로 판정된 경우에는 SV 비트를 1로 설정 하며 SE 경우에만 0으로 설정 한다. 따라서 캐쉬 읽기 시에 hit/miss를 결정하는 로직은 SEEB가 제공하는 SV 비트를 보고 1인 경우에만 hit를 발생 시키도록 구현 되어 있다. 그림 4에서와 같이 패리티 검사 error가 발생한 line에 대해서는 miss를 발생시키는 동시에 valid 비트는 0으로 설정되며PEN signal이 1로 설정 된다. PEN signal은 패리티 검사가 fail 되었음을 알리는signal이며 패리티 검사 결과 집계 시 fail 결과 이외의 중간 잡음이 쉬프트 레지스터로 들어가게 하지 않도록 하는 signal이다. 그리고 ERR signal은 패리티 check fail 결과를 쉬프트 레지스터에 저장하기 위하여 전달한다. 이렇게 전달된 신호들을 가지고 SEEB 블록은 error 분류 작업을 진행한다.

2.3.3 SEEB(Soft Error Evaluation Block)

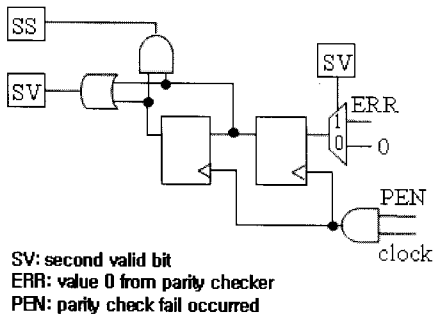

그림 5. SEEB의 구조
Fig 5. SEEB structure

그림5는 제안된 SEEB(Soft Error Evaluation Block)의 구조를 나타내고 있다. SEEB의 구성은 4개의 combinational 로직, 2단의 쉬프트 레지스터, SV 비트 그리고SS 비트(Soft Error Suspicious Bit)로 이루어져 있다. 기본적인 동작은 패리티 fail이 발생 되었을 때 결과를 받아들여 1비트씩 쉬프트 하는 것이다. 결과적으로 2번 패리티 검사 시 error가 발생하면 OR cell의 동작에 의하여 SV 비트를 0으로 설정 하여 해당 line이 soft error에 취약한 구조임을 표시한다. 그리고 한번SV 비트가 0이 되면 그림 5에서처럼 mux를 설정하여 패리티 결과가 쉬프트 레지스터 내부로 전파 되지 않도록 하였다. hit/miss 결정 로직이나 replacement counter는 SV 비트를 전달 받아 hit/miss 판단이나 linefill시에 활용한다. 결국은 SV 비트의 정보에 따라 해당 line이 사용 제한되도록 구현되어 있는 것이다. SEEB는 line의 soft error에 대한 취약성을 검사하기 위하여 제안 하였으며 동일 line이 두 번 이상 error가 발생한다면 해당 line이 구조적으로 결함이 있다고 판단하도록 구현 되어 있다. 기존 논문에서는 soft error에 의한 패리티 검사 error가 발생 하였을 때는 무조건miss를 발생시키고 해당 line을 무효화 시키거나 SV 비트를 두어 사용 제한시키는 방법을 사용하였다. 이 두 가지 방법은 모두 문제점을 가지고 있는데 무조건 해당line을 무효화 시킨다면 soft error에 취약한 line이 이후에도 계속 linefill되어 동일 error가 반복적으로 발생할 수 있는 가능성을 암시 하는 것이다. 그리고 error가 발생하였다고 해서 바로 사용제한을 하는 것도 MTTF가 비교적 긴 design으로 볼 때는 캐쉬의 사용면에서 비효율적인 방법이다. 따라서 해당 line이 soft error에 취약하다고 판단할 만한 근거를 모아 재 사용 여부를 결정 하는 것이 합리적인 접근 방법으로 생각된다. SEEB에서는 이전 발생하였던 패리티 검사 fail의 history를 종합하여 error가 두 번 이상 발생한 경우에만 해당 line을 사용 제한하도록 하여 최대한 line의 사용 효율을 높였다. 2.3.2절에서 언급된 바와 같이 본 논문에서는 soft error를 3가지로 분류하였다. 표 1에서와 같이 SF(soft error free), SS(soft error suspicious), SE(soft error occurred)가 그것인데, SF와 SS 의case는 각각 패리티 검사 error가 한 번도 발생하지 않거나 또는 1번 발생한 line이다.

표 1. 캐쉬 line의 상태 분류
Table 1. Category of cache line status

	SV		
SS	value	0	1
	0	00 (SE)	10 (SS)
	1	01 (N/A)	11 (SF)

SV, SS 두 개의 비트 조합이00(SE)인 경우가 최종적으로 사용 제한이 필요한 단계이며 11(SF)인 경우는 정상적인 line이다. 10(SS)인 경우가 판단 보류인 상황인데 특히 direct mapped 캐쉬인 경우에는 한번 soft error가 발생한 block이 사용 제한이 되면 다음 access부터는 매번 외부 메모리에서 data를 가져오게 되어 성능에 영향을 미치게 된다. 그러나 soft error가 발생한 line을 한 번 더 재사용 되도록 기회를 준다면 MTTF가 비교적 긴 design인 경우에는 비록 soft error에 취약한 구조를 가지고 있다 하더라도 error가 다시 발생할 때까지 정상적으로 사용 될 수 있는 것이다. 만약 SS case의 line이 그 이후로는 다른 원인의 soft error 공격으로부터 견디게 된다면 정상 line처럼 계속 사용 될 수 있는 것이다. 그림 6은 MTTF 변화에 따른 direct mapped 캐쉬의 hit rate 변화를 나타낸 것이다. 한 주기 동안의 hit rate가 99% 인 프로그램을 예로 들었으며 특정 line의 사용 제한으로 인하여 hit rate가 97%로 감소한다고 가정 하였다. 그리고 MTTF는 편의상 프로그램의 한 주기씩 증가 시켰다. 그림 6에서 알 수 있듯이 SS case의 cache line은 MTTF가 증가 할수록 평균 hit rate는 증가하며 error가 없는 수준의 hit rate에 도달 하는 것을 알 수 있다. 결함이 있는 line의 재사용 율이 증가하기 때문이다. 만약 SS로 설정된 line에서 연속적으로 error가 발생 했다면(SE case) hit rate는 최악이 되며 해당 line은 사용 제한이 될 것이다.

MTTF가 긴 design에서는 이런 방식으로 캐쉬 메모리의

사용 효율을 높일 수 있다. 따라서SEEB를 사용하게 되면 set associative 캐쉬의 오류를 복구할 수 있는 현재 구조의 장점을 살릴 수 있는 동시에 direct mapped 캐쉬의 효율적인 사용을 꾀할 수 있다. SS 비트에 대해서는 2.3.4절에서

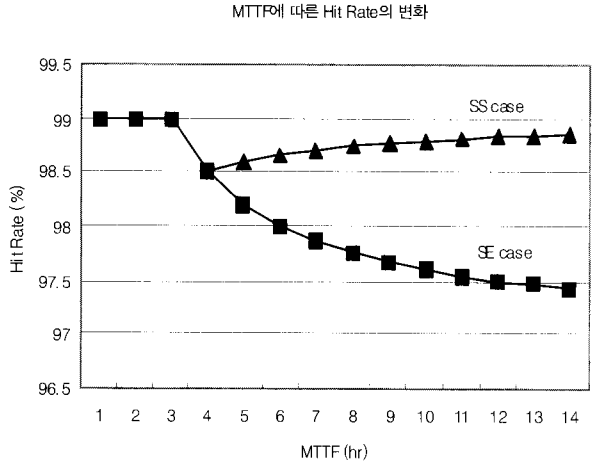

그림 6. MTTF 따른 hit rate 변화
Fig 6. Trend of hit rate affected by MTTF

2.3.4 Linefill시 faulty line의 처리

SEEB에 의해 SE(soft error 발생)로 처리된 line에는 다시 linefill이 시도 되어서는 안 되며 SS(soft error suspicious)인 경우에는 한번의 linefill기회가 더 주어져야 한다. 이러한 기능을 구현하려면 SEEB가 제공한 line 정보

표 2. Linefill way의 선정
Table 2. Selection of linefill way

case	SV#(3,2,1,0)	Linefill 가능 way
1	0000	None
2	0001	0
3	0010	1
4	0011	0/1
5	0100	2
6	0101	0/2
7	0110	1/2
8	0111	0/1/2
9	1000	3
10	1001	0/3
11	1010	1/3
12	1011	0/1/3
13	1100	2/3
14	1101	0/2/3
15	1110	1/2/3
16	1111	0/1/2/3

*SS bit 는 1 로 가정한다

와 연동하여 linefill을 수행하는 replacement algorithm이 요구된다. 이를 구현하기 위해서는 캐쉬 replacement algorithm을 수정하여야 하는데 캐쉬에는 두 가지 replacement algorithm이 존재한다. 하나는 순차적으로 eviction 될 way를 정하는 방법과 random 방식으로 정하는 방법이 있는데 어떤 방법을 사용하게 되던 SE로 분류된 line을 다시 linefill하지 않도록 구현하여야 한다. 본 논문에서는 round robin방식의 순차적 replacement algorithm을 기본으로 하여 수정한 방법을 구현하였다. SE로 분류된 line은 SV 비트가 0으로 설정 되므로 아래의 표2 과 같이 line fill 가능한 way를 정리한 표를 생각 해 볼 수 있다. 예를 들어 associativity가 4인 캐쉬를 가정해 볼 때, 4 way중에서 way0의 한 line 이 SV 비트가 0으로 설정 되었다고 가정하면 동일 index의 way0에 다시 linefill을 시도하면 안 된다. 따라서 way1, way2, way3중 한 곳에 linefill하도록 replacement counter를 설정하도록 하여야 한다. 이는 기존의 round robin algorithm에 예외 규정을 추가하여 로직으로 구현하였으며 software algorithm으로 볼 때는 hash구조의 linear probing을 구현 한 것과 비슷하다. 즉 현재 저장 장소에 저장이 허용되지 않을 때에는 counter를 1씩 linear하게 증가시키면서 다음 가능한 장소에 저장하는 것과 동일한 이치이다. 이 구현으로 way0에 SE로 분류된 line이 존재할 때 다음 linefill은 way1에 data를 linefill하게 된다. 두 번째 예로 way0를 access시 way0과 way2의 SV 비트가 0으로 설정이 되어 있다면 way1과 way3중에 한 곳을 linefill 위치로 선정하게 되는데 linear probing의 결과로 way1에 저장하도록 되어 있다. 최악의 경우에는 동일 index의 모든 way가 연속적으로 SV 비트가 0으로 설정 되어 있는 경우인데 이럴 경우는 해당 address를 access 할 경우에는 모두 외부 메모리를 access 하게 된다. 이 경우가 worst case의 soft error model 이며 가장 큰 캐쉬 성능의 저하를 가져 온다.

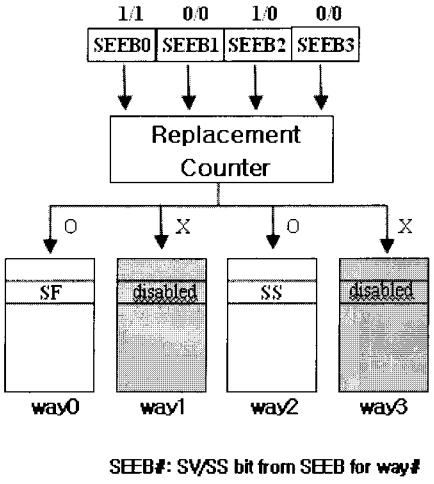

그림 7. Linefill시의 동작
Fig 7. Linefill operation

한 가지 더 고려해야 할 점은 SEEB에서 제공한 SS 비트를 활용하는 것이다. SS 비트는 이미 1번 패리티 검사에서 error,가 발생하여 soft error 요주의 line으로 분류 된 line이다. 따라서 linefill시에는 SF(soft error free)line 보다 우선순위를 낮게 책정하도록 하였다. 위의 두 번째 예를 고려해

볼 때 way0과 way0의 SV 비트가 0으로 설정 되어 있고 way1과 way3의 SS 비트가 각각 0과 1으로 설정 되어 있다면 way1보다는 way3이 linefill에 사용 된다는 것이다. 이러한 방법은 캐쉬 메모리 자원을 최대한 활용하기 위한 것이며 온전한 자원을 다 소모하였을 때 결함이 있는 자원을 최후에 사용하는 것이다. 그림 7은 지금까지의 설명을 구현한 block diagram이다.

지금까지 본 논문에서 구현된 캐쉬 구조에 대해서 설명하였다. 이와 같은 방법으로 faulty line에 대해서 선택적으로 사용을 제한시키고 새로운 way에data를 저장하는 방법을 사용하게 되면 동일error의 발생을 줄일 수 있다. 이 구조의 특징은 set associative 캐쉬와 더불어direct mapped 캐쉬 모두 soft error에 tolerance를 확보 할 수 있다는 것과 캐쉬 자원을 효율적으로 사용하여 기존 성능을 유지시킬 수 있는 방안이라는 점이다. 실험에서는 제안된 캐쉬 구조를 가지고function simulation, 성능 측정, area overhead 및 timing overhead 측정을 실시하였다.

3. 실 험 결 과

본 논문의 구현에 대한 실험은 상용 tool인 SimpleScalar 3v0d와 benchmark인 SPEC2000을 사용하였다. 256x32, 4way-set associative, write through mode 캐쉬를 사용하였고 SimpleScalar의 캐쉬 simulation model중 instruction 캐쉬를 본 논문의 구현대로 수정하여 simulation 수행 하였다.

3.1 Function simulation 결과

그림 8, 9는 SimpleScalar simulation tool인 sim-outorder의 trace 기능을 이용한 gzip 프로그램의 simulation log file이다. 특정 메모리 array의 비트를 임의로 조정하여 구현된 로직의 오류 복구 능력을 simulation 해 보았다. 결과를 보면 일반적인 캐쉬를 사용한 그림8은 soft error가 발생한 비트에 따라서 fail이 발생(invalid instruction exception)하는 반면 제안 된 로직을 구현한 그림9는 그림 8과 동일 line에서 fail이 발생하지않고 정상 동작함을 알 수 있다.

```
+ 263143  0x0040da30  0x000
* 263143  IF  0x00000001
+ 263144  0x0040da38  0x000
* 263144  IF  0x00000000
@ 72606
* 263143  DA  0x00000000
- 263143
* 263144  DA  0x00000000
@ 72607
* 263144  EX  0x00000000
@ 72608
* 263144  WB  0x00000000
```

그림 8. 기존 캐쉬 simulation 결과
Fig 8. Simulation log with existing cache

그림 8에서의 fail은 시스템의 오 동작 및 예외 상황을 야기 시키는데 흔히 볼 수 있는 갑작스런 시스템 reset 현상이나 data abort, prefetch abort같은 exception이 그것이다. 그림 9에서 유추 할 수 있는 상황은 패리티 검사 결과 fail 발생 하였고 SEEB에 의해서 해당 line이 요주의 line으로 변경 되었다는 것이다.(SV: 1, SS: 0) 하지만 단지 한 번 error가 발생하였으므로 사용 제한되지는 않았으며 replacement counter에 의해 동일 index의 다른 way를 사용하여 프로그램이 재 로딩 된 것을 유추 해 볼 수 있다. 이후 동일한 부분에서 다시 fail이 발생하면 해당 line은 사용 제한된다. 이러한 메커니즘이 본 구조의 특징이며 장점 이라 할 수 있겠다.

```
+ 263143  0x0040da30  0x000
* 263143  IF  0x00000001
+ 263144  0x0040da38  0x000
* 263144  IF  0x00000000
@ 72606
* 263143  DA  0x00000000
* 263144  DA  0x00000000
@ 72607
* 263143  EX  0x00000000
@ 72608
* 263143  WB  0x00000000
* 263144  EX  0x00000000
```

그림 9. 제안된 캐쉬의 simulation 결과
Fig 9. simulation log with proposed cache

3.2 성능 측정

그림 10, 11은 gzip 프로그램에 대한 SimpleScalar profile tool인 sim-outorder의 결과를 graph로 도시한 것이다. Frequency는 특정 address에 대한 access 빈도수를 나타내며 hit rate는 전체 access 중 miss rate를 제한 나머지를 나타낸다.

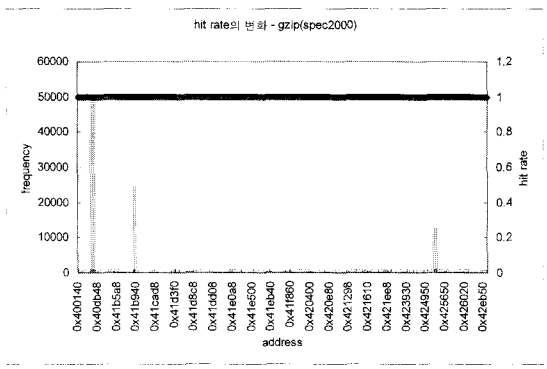

그림 10. 정상 동작하는 way가 있는 경우
Fig 10. Result with available way

그림 10은 동일 index에서 정상 동작 하는 way가 1개라도 있는 경우에 대해서 SPEC2000bench mark 중 gzip program을 수행한 결과이다. 그림에서 알 수 있듯이 error가 특정 몇 개의 way에서만 발생하거나 access 빈도가 낮은 address에서 발생하면 성능 저하 현상은 거의 없으며(hit rate가 거의 1에 가깝다) 기존의 성능을 유지 한다. 문제가 되는 것은 최악의 경우인데 동일 index의 모든 way가 패리티 검사에서 error가 발생 했을 경우이다. 이런 경우는 결국 해당 index의 캐쉬를 사용하지 못하고 모두 사용 제한이 된 상태이다. 이럴 경우 매번 외부 메모리에서 해당 index 내용

을 linefill 해야 하고 캐쉬 성능에 영향을 미치게 된다. 그림 11은 각 index별로 모든 way가 두 번 이상 패리티 검사 error가 발생한 것으로 가정하고 SPEC2000 bench mark를 수행한 결과 중 gzip 프로그램의 예를 보인 것이다. SEEB에 의해 동일 index의 모든 way가 사용 제한이 되었기 때문에 access가 많은 index를 수행 할 때는 hit rate가 저하되는 것을 볼 수 있다. 전체 SPEC2000 benchmark 프로그램의 결과를 볼 때 3% 이하의 hit rate 저하를 보인다.

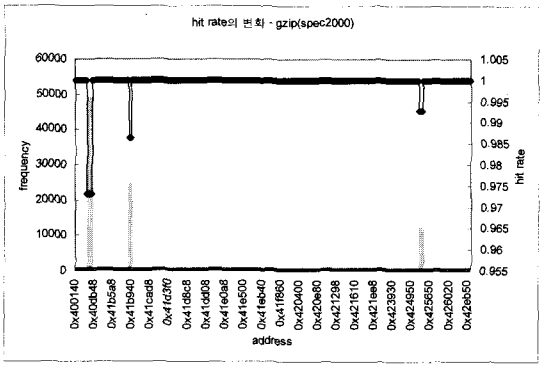

그림 11. 전체 way가 fail 인 경우(gzip)
Fig 11. Result with all way not available(gzip)

또한 MTTF에 따른 성능은 이미 그림 6에서 도시하였듯이 MTTF가 비교적 긴 design에서는 soft error가 없는 상태의 hit rate를 유지한다. 그림 11의 경우를 적용 시켜 본다면 SEEB가 구현된 경우와 그렇지 않은 경우에는 거의 3%의 hit rate의 차이를 보인다.

3.3 Area overhead

표3은 일반적인 캐쉬 구조에 대해서 제안된 각 구조 별 area 증가량을 표시한 것이다. SEEB 를 사용한 경우는 추가되는 SV, SS 비트가 flip-flop으로 이루어져 있기 때문에 패리티 비트 이외의 메모리 증가분은 없으며 line단위로 삽입이 되기 때문에 SEC-DED의 추가되는 패리티 비트보다 그 수가 적다. 따라서 area의 증가량은 기존의 패리티만을 사용하는 제안보다는 다소 크지만 단일 비트 오류 정정에 많이 쓰이는 SEC-DED 보다는 6.8%가량 작은 것을 알 수 있다

표 3. Block 별 area 증가량
Table 3. *Increase of area for each block*

	기존구조	SEEB 구조	SEC-DED
Logic	0.1	0.9	0.4
F/F	0	6	0
Memory	3	3	22
Total	1.3	2.7	9.5

단위:%

3.4 Timing overhead

$$T = h1\,t1 + (1-h1)\,h2t2 + (1-h1)(1-h2)\,tm. \qquad (1)$$

식 (1)은 메모리 access time을 나타내며 VLSI 시스템의 성능을 나타내는 척도로 사용된다. h1, t1,은 각각level1 캐쉬의 hit ratio및 access time을 나타내며 h2, t2는 level2 캐쉬의 hit ratio및 access time을 나타낸다. 그리고 tm은 main 메모리의 access time을 나타낸다. 그리고 각 level의 access time은 1, 3, 10의 unit time으로 가정하였고 패리티 검사에 의한 level1 캐쉬의 access time은 15% 증가하는 것으로 가정하였다.[16][6] 식 (1)을 가지고 가장 최악의 경우인 그림 11의 경우를 계산 해 보면 level1 캐쉬의 hit rate 저하에 따른 전체 시스템의 성능 저하를 확인 할 수 있다. 결국은 전체 way가 soft error가 발생 하였다고 가정한 경우 본 논문의 design에서는 최고20.8%의 timing overhead가 발생하는 것을 알 수 있다. 하지만 access 빈도가 낮은 address를 참조하거나 way가 한 개라도 정상인 address를 access하는 경우에는 hit rate의 변화가 거의 없으므로 15%대의 성능 저하가 발생함을 확인 할 수 있다. 이것은 패리티 검사를 추가함으로써 발생하는 overhead로써 일반적인 경우에 soft error는 단일 비트 error인 것을 감안 하면 SEEB에 의해 사용 제한된 line은 전체 시스템 성능에 큰 영향을 주지 않는 것을 알 수 있다. 패리티를 사용함으로써 발생하는 penalty는 일반적인 현상이며 대부분의 design에서 이러한 불이익을 감수하며 tolerance를 확보하고 있는 상황이다.[4] 하지만 통상 패리티 로직 만을 사용하는 방식이 timing 면에서는 ECC를 이용한 오류 정정 방식 보다 5% ~ 10% 성능이 좋다.[16][6] 이러한 penalty는 패리티 방식의 발전으로 개선될 수 있는 부분이기는 하지만 패리티 검사 성능의 향상은 area에 대한 overhead를 의미 하므로 성능과 area에 대한 trade off를 고려해야 하는 것은 설계자의 몫이다.

4. 결 론

본 논문은 메모리의 soft error를 대비한 캐쉬 구현 방법을 제안하였다. 특정 캐쉬 line에서 soft error가 발생하여도 시스템의 정지 없이 오류가 발생한 data를 복구하여 동작하며 error가 발생한 line에 대해서 선택적으로 사용을 제한함으로써 동일한 부분에서 error가 반복적으로 발생하는 것을 방지 할 수 있다. 특정 캐쉬 index에 대해서 일부 way가 사용 제한이 되더라도 나머지 way중 한 way라도 정상 동작하는 way가 있으면 캐쉬의 성능의 큰 저하 없이 data를 복구 할 수 있다. 그리고 MTTF가 큰 design인 경우에는 캐쉬 line의 사용 효율을 증가시켜 시스템의 성능을 향상 시켜 준다. 또한 기존의 오류정정 방식의(SEC-DED) 캐쉬보다 작은 area로 구현됨을 확인 하였다. 따라서 본 논문의 구현으로 캐쉬의 soft error에 대한 tolerance를 확보 하였고 향후 soft error에 대한 면역력을 보강하는 데 기여할 것으로 기대 된다.

참 고 문 헌

[1] S. G. Miremadi, H. R. Zarandi, "Reliability of protecting techniques used in fault-tolerant cache memories,"Electrical and Computer Engineering, 2005. Canadian Conference, pp:820-823, 1-4 May 2005.

[2] A. J. Tylka, W. F. Dietrich, P. R. Boberg, P.R.; E. C. Smith, J. H. , Jr. Adams, "Single event upsets caused by solar energetic heavy ions,"Nuclear Science, IEEE Transactions, Volume 43, Issue 6,Part 1,pp:2758 2766, Dec. 1996.

[3] E. Ibe, H. Kameyama, Y. Yahagi, H. Yamaguchi,"Single event effects as a reliability issue of IT infrastructure," Information Technology and Applications, 2005. ICITA 2005. Third International Conference, Volume 1,pp:555-560 vol.1, 4-7 July 2005.

[4] J. M. Benedetto, P. H. Eaton, D. G. Mavis, M. Gadlage, T. Turflinger,"Digital Single Event Transient Trends With Technology Node Scaling,"Nuclear Science, IEEE Transactions, Volume 53,Issue 6,Part 1,pp:3462-3465, Dec. 2006.

[5] J. F. Ziegler, et al., "IBM experiments in soft fails in computer electronics (1978-1994)," Terrestrial cosmic rays and soft errors, IBM Journal of Research and Development Volume 40, Number 1, pp. 3-18, 1998.

[6] O. A. Amusan, A. L. Steinberg, A. F. Witulski, B. L. Bhuva, J. D. Black, M. P. Baze, L. W. Massengill, "Single Event Upsets in a 130 nm Hardened Latch Design Due to Charge Sharing,"Physics Symposium, 2007. Proceedings. 45th Annual. IEEE International, Volume, Issue, pp:306-311, 15-19 April 2007.

[7] Vilas Sridharan, Hossein Asadi, Mehdi B. Tahoori,David Kaeli, "Reducing Data Cache Susceptibility to Soft Errors,"Dependable and Secure Computing, IEEE Transactions, Volume 3, Issue 4, pp:353-364, Oct.-Dec. 2006.

[8] L. Lantz II,"Soft errors induced by alpha particles," Reliability, IEEE Transactions on Volume 45, Issue 2, pp. 174-179, June 1996.

[9] T. Calin, et al., "Topology-Related Upset Mechanisms in Design Hardened Storage Cells," Radiation and Its Effects on Components and Systems, 1997, RADECS 97.

[10] G. S. Sohi, "Cache Memory Organization to Enhance the Yield of High-Performance VLSI Processors," IEEE Trans. Comp., Vol. 38, No. 4, pp. 484-492, April 1989.

[11] P. P. Shirvani, P.P., E.J. McCluskey, "PADded cache: a new fault-tolerance technique for cache memories," VLSI Test Symposium, 1999. Proceedings. 17th IEEE, pp. 440-445, 25-29 April 1999.

[12] S. J. Walsh, J. A. Board, "Pollution control caching" Computer Design: VLSI in Computers and Processors,"1995. ICCD '95. Proceedings., 1995 IEEE International Conference, pp:300 306, 2-4 Oct. 1995.

[13] S. Ozdemir, D. Sinha, G. Memik, J. Adams, H. Zhou, "Yield-Aware Cache Architectures Microarchitecture," 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium, pp:15-25, Dec. 2006

[14] Hsin-Chuan Chen, Jen-Shiun Chiang "Design of an adjustable-way set-associative cache," Communications, Computers and signal Processing, 2001. PACRIM. 2001 IEEE Pacific Rim Conference, Volume 1,pp:315-318 vol.1, 26-28 Aug. 2001

[15] C. W. Slayman, "Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations,"Device and Materials Reliability, IEEE Transactions, Volume 5, Issue 3, pp: 397-404, Sept. 2005.

[16] Richard Phelan, "Addressing Soft Errors in ARM Core-based Designs, Improving fault tolerance through error detection and correction," Whitepaper, www.arm.com, December 2003.

[17] A. J. smith, "Cache memories," ACM Comput. Surveys, Vol. 14, pp: 473-530, Sept. 1982.

[18] S. S. Mukherjee, J. Emer, S. K. Reinhardt, "The soft error problem: an architectural perspective,"High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium, Volume, Issue, pp: 243247, 12-16 Feb. 2005.

[19] D. A. Patterson, et al., "Architecture of a VLSI Cache for a RISC," Proc. Int'l Symp. Comp. Architecture, Vol. 11, No. 3, pp. 108-116, June 1983.

[20] X. Luo, and J.C. Muzio, "A Fault-Tolerant Multiprocessor Cache Memory," Proc. IEEE Workshop on Memory Technology, Design and Testing, pp. 52-57, August 1994.

[21] M. Mehrara, M. Attariyan, S. Shyam, K. Constantinides, V. Bertacco, T. Austin,"Low-Cost Protection for SER Upsets and Silicon Defects," Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE '07, pp. 1-6, 16-20 April 2007.

[22] Allan. H. Johnston, "Scaling and Technology Issues for Soft Error Rates," 4th Annual Research Conference on Reliability, pp. 1-9, Stanford University, October 2000.

[23] Nicholas J. Wang, Sanjay J. Patel, "ReStore: Symptom-Based Soft Error Detection in Microprocessors," Dependable and Secure Computing, IEEE Transactions on, Volume 3, Issue 3, pp. 188 201, July-Sept. 2006.

[24] P. M. Carter, B. R. Wilkins, "Influences on soft error rates in static RAMs," Solid-State Circuits, IEEE Journal of Volume 22, Issue 3, pp. 430436, Jun 1987.

[25] S. S. Mukherjee, J. Emer, T. Fossum, S. K. Reinhardt,"Cache scrubbing in microprocessors: myth or necessity?," Dependable Computing, 2004.

Proceedings. 10th IEEE Pacific Rim International Symposium, pp:37-42, 3-5 March 2004.

[26] S. G. Miremadi, H. R. Zarandi, "Reliability of protecting techniques used in fault-tolerant cache memories," Electrical and Computer Engineering, 2005. Canadian Conference, pp:820-823, 1-4 May 2005.

저 자 소 개

이 종 호 (李 鍾 豪)

1972년 8월 1일생. 1998년 광운대학교 전자공학과 졸업.
1998년~현재 삼성전자 SOC 연구소 Processor Architecture Lab. 책임 연구원
Tel : 031-209-2867
E-mail : json.lee@samsung.com

조 준 동 (趙 浚 東)

1957년 7월 21일생. 1980년 성균관대학교 전자공학과 졸업
1983년~1987년 삼성전자 Memory Device 사업부, CAD Team
1989년 Polytechnic Univ. Brooklyn 졸업 (공학석사)
1993년 Northwestern Univ. Evanston 졸업(공학박사)
1993년~1995년 삼성전자 Micro Device 사업부, CAD Team
2000년~2001년 Visiting Scientist, Design Automation IBM T.J. Watson Research Center
1995년~현재 성균관대학교 정보통신공학부 교수
Tel : 031-209-7127
E-mail : jdcho@skku.ac.kr

표 정 렬 (表 正 烈)

1967년 5월 14일생. 1990년 한양대학교 전자공학과 졸업.
1992년 동대학 전자공학과 졸업(공학석사)
1992년~현재 삼성전자 SOC 연구소 Processor Architecture Lab. 수석 연구원
Tel : 031-209-2958
E-mail : jrpyo@samsung.com

박 기 호 (朴 基 豪)

1969년 8월 11일 생. 1993년 연세대학교 이과대학 전산과학과 졸업
1995년 연세대학교 컴퓨터과학과 졸업(이학석사)
2000년 연세대학교 컴퓨터과학과 졸업(공학박사)
2000년~2001년 University of Utah Post-Doctoral Research Associate
2002년~현재 삼성전자 SOC 연구소 Processor Architecture Lab. 책임 연구원
Tel : 031-209-8491
E-mail : giho.park@samsung.com